

本周周报（6.2-6.8）

刘昊南

本周工作

1. 跟海东师兄讨论了 DTI 项目中 Fiber DB 的重构细节
2. Fiber DB 是按 cell-hash 的方式构成的，把空间划分为规则的 cell，每个 cell 中存在一个哈希表。在原来的设计中，哈希表的 key 是 Fiber Model ID+Fiber ID，能够唯一确定一条 fiber；value 是这条 fiber 在 cell 中的节点的三维空间位置，以及两个编码了上一个节点和下一个节点所在 cell 的值。但是，在之前的实验中，哈希表的内存开销过大，甚至导致我的机器内存会爆掉的情况，所以需要重新设计哈希表的 key 和 value。
3. 在原来的设计中，value 中的三维空间位置是 3 个 float 值，编码上一个节点和下一个节点的是两个 char 值，每个 value 消耗 14 个字节。在新的设计中，我们不将三维空间位置的值放入哈希表，这些值保存在硬盘上的 page 文件中，哈希表的 value 值仅保存一个指向三维空间位置所在那个文件的那个位置的指针。另外，编码 fiber 经过的上一个 cell 或下一个 cell 并不需要一个字节，因为跟一个 cell 相邻的 cell 只有 26 (9+8+9) 个，5 个 bit 已经绰绰有余。
4. 经过认真的考虑，方案如下：
 - a) Last Cell 和 Next Cell 各用 5 个 bit 来编码
 - b) 指向 page 文件的 Page ID 用 8 个 bit 编码，这样最多可以有 256 个 page 文件
 - c) Fiber 的起始点在 page 文件中的位置用 34 个 bit 编码，这样每个 page 文件最大可以超过 16G
 - d) 节点的三维空间坐标在 Fiber 内部的偏移位置用 12 个 bit 编码，这样单个 fiber 最多可以有 4096 个节点，足够使用按照上面的设计，一个 value 只需要占用 8 个字节 (5+5+8+34+12=64bit)，节省了 6 个字节的空间
5. 我们先将所有的 fiber 在空间上进行聚类，属于相同 cluster 的 fiber 存储在同一个 page 文件中。Page 文件实际上就是按 fiber 进行存储，fiber 内部按从起始节点到终止节点的顺序存储节点的三维空间坐标。对于 page 文件我们可以采用 LRU 等交换策略来载入内存，空间相近的 fiber 放在同一个 page 中，所以也不会出现频繁换页。而且，这种方式方便了对于单条 fiber 的遍历，只需要找到 fiber 在 page 文件中的偏移，就可以将起始点到终止点直接读入，不然遍历每个点都得做一次 hash 查找。
6. 对于保存哈希表的 cell 文件，也可以进行优化，在原来的设计中，将空间中的所有 cell 分成了几个 block，每个 block 有 32x32x32 个 cell。每个 block 存

成一个文件，当需要某个 cell 时，判断它在哪个 block，若 block 已经在内存中，可以直接读取，否则将整个 block 从内存中读入。这种划分导致有的 block 很小，有的 block 很大，因为每个 cell 包含的点的数目大小不一。而且，为了单个 cell 而将整个 block 读入代价太大，很多 cell 可能根本用不上。另外，当 Fiber Model 很多时，会产生一些很大的 block，比如当有 50 个 fiber model 时，最大的 block 有 600 多 MB。

7. 对于 cell 文件，我的想法是将所有 cell 放入同一个文件中，这样或许会产生一个巨大的文件，但是没关系。另外再建立一个针对 cells 文件的索引文件，其中的每一条索引 cell 在文件中的偏移量，索引的每一条长度都一样，而且按先 x 后 y 再 z 的方式排列，定位迅速，读取高效。程序中再设立一个固定大小 cells 的队列，采用 LRU 的置换方式。这样，能够快速定位、读取某个 cell 里的哈希表，也能缓存一些常用的 cell，又不会产生大的内存浪费导致内存爆掉。

下周计划

1. 目前已经完成了哈希表的重构，下一步即将完成 page 文件的生成以及 cells 文件及其索引文件的生成。